

20
150 pts.

ARUN

Enciclopedia Práctica del Spectrum



Nueva Lente/Ingelek



COLOR EN NUESTRO T. V.



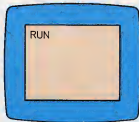
El color es uno de los atributos que hacen del Spectrum un ordenador atractivo y versátil, manteniendo una extraordinaria sencillez de manejo.

Hasta ahora no sabemos hacer uso de esta importante herramienta, pero en el transcurso de este capítulo y el siguiente, perfilaremos los conocimientos necesarios como para empezar a explorar las posibilidades cromáticas de nuestro aparato, sacándole partido a esta nueva e importante faceta del mismo.

Lo primero que debemos saber sobre este tema es que, a efectos de color, la pantalla se encuentra dividida en dos zonas bien diferenciadas, cuya denominación podría ser «primer» y «segundo» plano, o zona «circundante» y zona «central» de la pantalla.

La zona de la pantalla sobre la cual podemos escribir, es una porción rectangular de 26 filas por 32 columnas. El resto de la misma, es decir, la parte que se encuentra por encima y por debajo, o a la izquierda y derecha de los márgenes citados, no es accesible a través de la sentencia PRINT.

La pantalla que se presenta en el ordenador puede dividirse en dos zonas, exterior (no accesible por PRINT) e interior.



ZONA ACCESIBLE POR PRINT

ZONA NO ACCESIBLE POR PRINT



A partir de este capítulo, podremos capturar a color personajes a ojos de los creadores de nuestro Spectrum de generación de color.

Esta zona no utilizable para la escritura, contiene un espacio de «fondo» o segundo plano, por donde la pantalla, y puede adquirir cualquiera de los ocho colores básicos de que dispone el Spectrum, resaltados en la parte superior de los teclados correspondientes a los números. Como podemos observar en la tabla adjunta, a cada uno de los colores le corresponde un número determinado, el cual utilizaremos para referirnos a él cuando programemos nuestro ordenador. Así por ejemplo, diremos que vamos a escribir en tinta 0, en vez de en tinta negra, o que vamos a emplear el color 2, en vez del color rojo, etc.

Lo mismo sucede con la parte central de la pantalla, con la ventaja de poder acceder a un segundo juego de colores, basado en el ya descrito. Este segundo juego de ocho colores se compone a partir de los básicos más un atributo denominado BRELLO. De esta forma, podemos decir que los ocho colores básicos son oscuros y los ocho restantes de tono más claro (más brillantes). Con esto hemos hablado de los dos colores de fondo posibles, el del marco de la pantalla, y el de la zona accesible por PRINT. Dentro de esta última, podemos distinguir aún dos sub-planos, conformando algo así como un primer plano y un «primerísimo» plano, siendo éste último el color en el cual escribimos realmente.

Llegados a este punto, quizá veamos las cosas

!

En la pantalla se pueden definir dos zonas: la exterior, denominada marco, y en la cual no se pueden escribir caracteres; y la interior, donde se realizan las impresiones mediante PRINT.

*

Cada uno de los colores recibe un código numérico del 0 al 7.

*

BORDER altera el color del marco de la pantalla de forma inmediata.



La pantalla del Spectrum puede considerarse similar a una mesa (marco), en la cual sus superficies para escribir (verver términos) sobre un papel (fondo).

más claras con un ejemplo, bastante descriptivo además, de los nombres de las sentencias BASIC que gestionan el color en el Spectrum.

Supongamos que nos disponemos a escribir una carta de forma manual. Para ello, nos proveemos de una mesa donde escribir, papel y bolígrafo. Por comparación de conceptos, podemos decir que el color de la mesa es el del marco de la pantalla, donde no podemos escribir, el del papel es el de la zona imprimible, y por último, el de la tinta del bolígrafo es el color del primerísimo plano anteriormente citado.

La sentencia BORDER nos permite dar color al marco de la pantalla.



LA SENTENCIA BORDER

Por medio de la sentencia **BORDER**, seguida de un código de color básico, es posible cambiar el tono del marco de la pantalla, el cual, al encendido del ordenador, adopta el blanco.

La gama de colores de este marco de pantalla, está restringida al ámbito de los ocho fundamentales. Veámos algunos ejemplos de los efectos que podemos conseguir, con su empleo, comenzando antes de cada ejercicio por teclear **BORDER 7** (marco blanco) para restablecer el color inicial. **FOR 1-0 TO 7: BORDER 1: PAUSE 0: NEXT 1**



También es posible obtener marcos con franjas múltiples.

Esta línea nos muestra, por cada pulsación de una tecla, la relación ordenada de colores disponibles, desde el más oscuro hasta el más claro. Otra forma de ver el color del marco de la pantalla, es el conseguido alterando dato a gran velocidad: **10 FOR 1-0 TO 7: BORDER 1: NEXT 1: GO TO 10**.

Finalmente, he aquí un último modo de obtener efectos en esta zona de la pantalla, aunque mucho más emborullados: **10 BORDER 2: BORDER 3: BORDER 4: BORDER 5: BORDER 6: GO TO 10**. Si este efecto de bandas de color no nos parece suficientemente sorprendente, podemos



pretor a pulsar cualquier tecla, para verlo desplazarse de arriba abajo por la pantalla.

Estos efectos de color no se emplean muy a menudo, puesto que requieren una atención permanente del ordenador en esta tarea, lo cual resulta lógicamente bastante inútil. En todo caso, pueden ser empleados en combinación con otros sistemas BASIC que estudiaremos más adelante, para amenazar algunos tiempos de espera, por ejemplo, aquellos transcurridos desde la edición de una pantalla de instrucciones de un juego, hasta que la hemos leído y pulsamos una tecla para evitar esta circunstancia.

Sin duda, lo más frecuente es utilizar la sentencia **BORDER** del modo más simple estudiado al principio de este epígrafe. De esta forma, una vez que hemos asignado un determinado color al marco de la pantalla, éste permanece así hasta que indicamos lo contrario mediante una nueva sentencia **BORDER**. Explicaremos ahora brevemente el por qué del marco a franjas que hemos visto aparecer, por lo que puede ilustrarnos sobre el funcionamiento interno del ordenador en relación con el color del marco.

Cuando el televisor recibe la señal de video del ordenador, comienza su impresión en la pantalla horizontalmente, de arriba a abajo, de una manera cíclica y constante. A este fenómeno se lo conoce bajo el nombre de **BARRIDO**. El color del marco no es más que uno de las señales que el ordenador envía al televisor, y así, si alteramos dicha señal de una forma lo suficientemente rápida, conseguiremos que al comienzo del barrido el marco sea de un color, un lapso de tiempo más tarde, antes de que finalice el barrido, de otro, y así sucesivamente hasta generar las franjas que hemos apreciado.

De hecho, si prestamos atención a lo que sucede en el marco de la pantalla cuando leemos o gra-



En el proceso de formación de la imagen en el televisor, se produce un barrido horizontal cíclico de la pantalla.

bamos un programa en cinta, observaremos que multitud de franjas ocupan esta zona de la pantalla. Ello es debido a que el ordenador envía, al tiempo que lee y escribe, diversas señales al televisor, y puesto que es capaz de hacerlo a una velocidad mayor que la que nosotros conseguimos con **BORDER** desde el BASIC, puede generar un mayor número de franjas.

Aunque en principio el marco de la pantalla sea una zona poco interesante, dado que en ella no podemos escribir, no debemos desoír la posibilidad de su manejo. En el quinto capítulo de la sección de **TU SPECTRUM**, encontraremos más información sobre este tema.

Si en un corto espacio de tiempo (inferior a un barrido), alteramos varias señales diferentes como color de marco por la pantalla, este trabajo se realiza sucesivamente al correspondiente, produciendo un marco a franjas.



I



II



III



IV



V



VI

!

Para que **PAPER** o **INK** afecte simultáneamente a toda la pantalla, hemos de especificar a continuación un comando **CLS**.

*

El brillo suplementario que se obtiene mediante la sentencia **BRIGHT** (en inglés, *brillig*), no afecta al marco de la pantalla, sino sólo a su zona interior.

*

Los atributos **BRIGHT** y **FLASH** (brillo e intermitencia), afectan simultáneamente a los **INK** (brillo) y el **PAPER** (brillo) de un carácter.

*

Sólo **BORDER** tiene un efecto inmediato sobre la pantalla, el resto de los atributos de color afectan a los caracteres escritos a continuación, pero no a los ya presentes en la pantalla.

!

Al cambiar el color del marco al estado original de los atributos de color se el siguiente **BORDER 7, PAPER 7, INK 0, BRIGHT 0 y FLASH 0**

*

Se alteramos la salida de color cuando el marco es un tiempo inferior al de un línea; podemos obtener marcos con franjas multicolor

LA SENTENCIA PAPER

Del mismo modo que **BORDER** asigna color al marco de la pantalla, **PAPER** lo hace para el centro de la misma. Sin embargo, la sentencia **PAPER** es mucho más potente: permite asignar un color determinado a cada posición concreta de la pantalla, es decir, a cada carácter en particular, en dos modalidades de brillo, gracias a la combinación con la sentencia **BASIC BRIGHT**, estado más adelante.

La estructura de la sentencia **PAPER** es similar en todo a **BORDER**, y va seguida por tanto de un código de color. En el siguiente ejemplo, veremos las posibles combinaciones de colores de marco y fondo, empleando los fundamentales: **FOR I=0 TO 7: FOR J=0 TO 7: BORDER I: PAPER J, CLS: NEXT J, NEXT I**

R U N

Las combinaciones de PAPER, BRIGHT se pueden utilizar en cada carácter, independientemente del resto de la pantalla.

Conviene que fijemos la atención en la sentencia **CLS** que sigue a **PAPER**. Este **CLS**, produce el efecto de esbucar todos los colores de fondo de la zona de escritura de caracteres, en el valor del último código indicado en una sentencia **PAPER** genérica. De esta forma, conseguimos la uniformidad de color de toda la pantalla imprimible, sin tener que recurrir a colorear, castillo a castillo, las 768 (24 filas de 32 columnas), que componen la pantalla.

Ael pues, pese a tener una sintaxis muy parecida, las sentencias **BORDER** y **PAPER** difieren mucho en su comportamiento, puesto que mientras **BORDER** asigna el color a todo el marco de la pantalla, **PAPER** no produce un efecto inmediato aparente. Veámoslo con un ejemplo. Si apagamos y encendemos el ordenador, apreciaremos que tanto el marco como el fondo de la pantalla interior son blancos. A continuación ejecutamos una instrucción **BORDER 2**, y el marco de la pantalla pasará a ser de color rojo, de for-

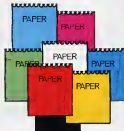


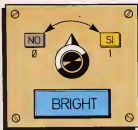
La extraordinaria velocidad del código máquina permite generar más franjas que las obtenidas desde el BASIC mediante BORDER. Esto es lo que sucede cuando muestra Spectras lo o prueba programo.

má inmediata. Ahora ejecutemos **PAPER 2** el mensaje **OK** se editará en la última línea de la pantalla, pero sin embargo el interior de la pantalla sigue siendo blanco (¿qué es lo que ha ocurrido? ¿caso la instrucción no ha sido formulada correctamente?)

En absoluto. El problema estriba en que el efecto de **PAPER** no es inmediato, como en el caso de **BORDER**, sino que afecta a las instrucciones de

Las sentencias PAPER nos permiten asignar el color de fondo de los caracteres a escribir en la pantalla.





Muestra **BRIGHT** para *oscurecer o aclarar el modo de brillo suplementario*

manejo de pantalla sucesivas, como **PRINT**, **TAB**, **CLS**, etc. - Probermos a escribir una letra **A** en la pantalla. **PRINT "A"**. Por fin hace su efecto la sentencia **PAPER**: la letra ha sido escrita sobre un fondo rojo.

En resumen, podemos afirmar que la sentencia **BORDER** indica al Spectrum que debe alterar el color de todo el marco de la pantalla de forma inmediata; mientras que **PAPER** lo señala el color que debe emplear como fondo en las próximas operaciones de escritura.

Este hecho nos ha ayudado a descubrir también que el efecto de **CLS** no es sólo el de borrar la pantalla, como habíamos visto hasta ahora, sino que también establece como color de fondo para toda la pantalla el seleccionado con la última sentencia **PAPER** (blanco inicialmente, al encender el ordenador).

Como ya hemos dicho, la sentencia **BRIGHT** es un complemento opcional de **PAPER**, que permite dar brillo al color. Veamos una combinación de sentencias **PAPER** y **BRIGHT**, para conseguir una «carta de ajuste» de color de fondo:

```
10 REM - AJUSTE DE COLOR
20 FOR I=0 TO 1
30 BRIGHT I
40 FOR J=1 TO 10
50 FOR K=0 TO 7
60 PAPER K:PRINT " "
70 NEXT K
80 NEXT J
90 NEXT I
100 BRIGHT 0
```

Como podemos apreciar en el listado, la sentencia **BRIGHT** lleva como parámetro un número que adopta los valores 0 ó 1. El primero se utiliza para indicar que el color se encuentra en el

modo normal, y el segundo que posee brillo suplementario. Como en el caso de **PAPER**, las instrucciones **BRIGHT 0** y **BRIGHT 1**, tendrán efecto sobre las escrituras sucesivas, pero no sobre toda la pantalla.

Finalmente, resta advertir que si utilizamos para la instrucción **BRIGHT** un parámetro menor que cero, obtendremos el mensaje de error **B illegal out of range** (entero fuera de rango), por el contrario, empleando parámetros por encima de 1, el mensaje emitido será **K invalid colour** (código de color no válido). Una excepción a esto último será el código ocho, cuyo peculiar efecto estudiaremos posteriormente.

LA SENTENCIA INK

Hasta ahora hemos visto cómo dar color a la mesa sobre la que nos apoyamos (**BORDER**), y al papel en el cual escribimos (**PAPER**), por otro, nos falta conocer la manera de depositar la tinta adecuada en nuestro bolígrafo.

A través de la sentencia **INK** se define el color

*La sentencia **INK** nos permite alterar el color de primer litro de las consuetas o corchetas.*



!

Los parámetros para **BRIGHT** (brillo) y **FLASH** deben estar comprendidos entre 0 y 1, significando dos corchetas y trescientos del modo, respectivamente.

*

Las diferencias entre los tonos con y sin brillo suplementario de un mismo color de consuetas según se consuetas éste.

de cada tinta. De forma similar a casos anteriores, la sentencia va seguida de un parámetro que señala el código de color deseado. Como era de esperar, tanto su sintaxis como su funcionamiento es muy similar a PAPER.

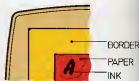
Así pues, el cambio del color de tinta no afecta de forma inmediata a toda la pantalla, sino sólo a las impresiones que se efectúan a partir de la ejecución de la instrucción correspondiente. Esto nos permite escribir en cada posición de carácter (coordenada fila-columna), con el color de papel (en inglés paper) y tinta (en inglés ink) que deseamos, independientemente de los que se utilizan para las otras posiciones de la pantalla. Aprovechando parte del programa anterior, podemos escribir ahora uno que nos muestre una carta de ajustes completa, de combinaciones de colores de papel y tinta, con y sin brillo.

```

10 BIR = AJUSTE DE COLOR
20 FOR I=0 TO 1
30 BRIGHT I
40 FOR J=1 TO 10
50 FOR K=0 TO 7
60 PAPER K:PRINT " "
70 NEXT K
80 NEXT J
90 NEXT I
100 FOR I=0 TO 1
110 BRIGHT I:PAPER 7
120 FOR J=1 TO 2
130 FOR K=0 TO 7
140 NEXT K
150 PAPER 0
160 FOR J=3 TO 7
170 FOR K=0 TO 7
180 NEXT K
190 NEXT J
200 PAPER 7: BRIGHT 1:INK 0

```

Tanto para PAPER como para INK, el rango de su parámetro está entre cero y siete, con dos niveles de brillo.



Los colores de BORDER, PAPER e INK son totalmente independientes entre sí.

Por último, y en relación con el margen en el cual se mueven los parámetros para INK y PAPER, éstos no deben ser inferiores a cero, puesto que como en el caso de BRIGHT, se producirá un error del tipo B integer out of range (lítera fuera de rango).

En esta ocasión, los colores admitidos llegan hasta el siete, aunque a ellos hemos de añadir dos códigos más: el ocho y el nueve, cuyo efecto no es exactamente el de un color particular, sino algo más especial que estudiaremos en el próximo capítulo. Cualquier parámetro por encima de nueve, provocará la emisión del mensaje de error K invalid colors, cuyo significado hemos tenido la oportunidad de conocer anteriormente.

CONSIDERACIONES GLOBALES

Si somos un poco observadores, veremos que el orden de los códigos de color se corresponde con una secuencia desde los más oscuros hacia los más claros. De hecho, si anulamos por un momento el color de nuestra T V, veremos que aparecen diferentes tonalidades de grises, ordenados de menor a mayor luminosidad.

Del mismo modo, se hace patente que la diferencia entre los colores con y sin BRIGHT es muy acentuada en los tonos claros, y casi imperceptible en los oscuros. Este efecto debe ser tenido muy en cuenta, puesto que puede ser aprovechado a nuestro favor en alguna ocasión, dentro de este mismo epígrafe, cuando tratemos otra de las características de BRIGHT, nos daremos cuenta del por qué de esta afirmación.

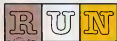
Puesto que el marco de la pantalla sólo puede re-

presentar los ocho colores básicos, y no se ve afectado por el factor de brillo, podemos utilizarlo como referencia sobre la luminosidad de los colores empleados en la pantalla interior. Ello nos conducirá a curiosos efectos ópticos, como el que a continuación describimos.

Ya hemos comprobado anteriormente que cuando encendemos el ordenador, el color inicial para marco y fondo de pantalla es el blanco, encontrándose ambos en la modalidad de brillo normal. Pues bien, ejecutemos la siguiente instrucción **BRIGHT 1:CLS**. Gracias a ella, conoceremos el modo de brillo suplementario, y borraremos la

CODIGOS DE COLOR

0	NEGRO
1	AZUL OSCURO
2	ROJO
3	ROSA
4	VERDE
5	AZUL CLARO
6	AMARILLO
7	BLANCO



Para mejorar el brillo suplementario de fondo, sin que aparentemente afecte al papel, debemos emplear fondos oscuros.

pantalla, de forma que el nuevo brillo afecte completamente el interior de la misma.

Casi como por arte de magia aproximamos que el cambio no se ha efectuado en el interior de la pantalla, como nosotros esperábamos, dado que sabemos que **BRIGHT** no afecta al marco. Aparentemente, el fondo ha permanecido en su color blanco inicial, y el marco de la pantalla ha adquirido un tono gris, que en teoría se encuentra fuera de la gama del Spectrum, y más aún, de las posibilidades de esa zona de la pantalla. En realidad, todo ha sido un simple efecto óptico. Como nosotros ya suponíamos, **BRIGHT** no ha afectado al marco, sino al fondo, pero el contraste del brillo suplementario entre uno y otro blanco, nos ha hecho creer que al primero de los colores, no es tal blanco, sino un perfecto gris. Este efecto óptico no debe ser desaprovechado, y combinado con inteligencia, puede arrancarnos al Spectrum algún color más que aquellos de los cuales originalmente dispone, o al menos dar tal impresión.

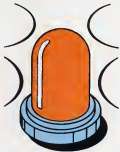
Como hemos dicho unos párrafos más atrás, esta diferencia entre los tonos brillantes y no brillantes, decimos según se ve oscureciendo la gama, y así como en el blanco está escaturida, he-

cho, este que hemos podido comprobar tan solo hace unos instantes, en el negro es prácticamente imperceptible.

El siguiente mini-programa nos ayudará a comprobar esta escala decreciente de diferencias, gracias al contraste entre el color del marco (sin brillo), y el del interior (con brillo suplementario), pulsemos una tecla por cada vez que queramos pasar al próximo color: **BRIGHT 1:FOR I=7 TO 0 STEP -1: BORDER 1: PAPER 1:CLS: PAUSE 0: NEXT I.**

Otra característica a considerar, en lo referente al atributo de color **BRIGHT**, es que afecta tanto al papel como a la tinta que concierne en una determinada posición de la pantalla. Como ya vimos anteriormente, en una misma posición de la pantalla (coordenada fila-columna), podemos hacer coexistir un color diferente de la tinta y del papel; sin embargo, si decidimos que dicho posi-

La rotación FLASH nos permite crear caracteres con colores intermitentes.



i!

Los colores de fondo y primer término que se otorgan a un carácter, así como los otros atributos (**BRIGHT** y **FLASH**) afectan a cada carácter de la pantalla, independientemente del resto de ellos y del marco de la misma.

*

El mensaje **B** borra el contenido fuera de rango (dentro de cuando queremos utilizar valores negativos para los atributos de color.

*

El mensaje **K** invierte el color (código de color no válido) espacio cuando utilizamos como penúltimo para los atributos de color códigos que superan el número.

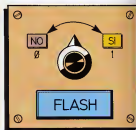
ción se encuentre con brillo suplementario, este hecho afectará tanto al color de la tinta como del papel en esa posición. Una vez que hemos visto esto, estamos en condiciones de explicar por qué digmos que le podríamos sacar provecho a la diferencia decreciente entre los tonos brillantes y no brillantes de los diversos colores. Supongamos que queremos escribir la letra inicial de una palabra en tono brillante, para destacarla del resto de la misma, pero sin embargo, nos interesa que permanezca de igual tono el fondo, por ejemplo, porque lo hemos hecho coincidir con el color del marco, para así aumentar las dimensiones aparentes de la pantalla (capítulo cinco, sección de TU SPECTRUM). Esta opción es aparentemente imposible, dado que el brillo afectará tanto a la tinta como al fondo (papel) del carácter.

La solución se encuentra en no utilizar en el programa un color de fondo claro, sino todo lo contrario: el color negro, o en su defecto el azul oscuro. Gracias a ello, aunque la cualidad de brillo suplementario existe también al fondo, su efecto es prácticamente imperceptible, y no es en la tinta, para la cual habremos empleado un tono muy claro, a ser posible el blanco.

UN ULTIMO TRUCO

Pero además de las posibilidades de color descritas, contamos con una más, quizá la más espectacular de todas: la intermitencia. Dado que en algunas ocasiones se hace conveniente resaltar de forma especial un determinado texto, el Spec-

FLASH produce una alternancia cíclica entre los colores de fondo (**PAPER**) y primer trazo (**INK**) de un carácter.



Los parámetros para FLASH están comprendidos entre 0 y 1, estable a voluntad.

trum incorpora además de las características de brillo (**BRIGHT**), la de intermitencia, obtenida mediante la sentencia **FLASH**, seguida de los parámetros 1 ó 0, para conexión o desconexión del modo, de forma idéntica a como funcionamos con **BRIGHT**.

Los errores que se producen al no respetar estos parámetros son los mismos que los emitidos con **BRIGHT**, incluyendo la posibilidad de utilizar el código especial 8, que será objeto de estudio en el próximo capítulo.

El efecto de **FLASH** es producir una alternancia intermitente entre los colores de tinta (**INK**) y fondo (**PAPER**) de un carácter, de forma automática, y con una velocidad constante. Con este nuevo aditamento, son muchas las combinaciones de color posibles dentro del área accesible de la pantalla; podríamos decir que casi infinitas. De hecho, tenemos que considerar lo siguiente:

- PAPER:** Valores de 0 a 7
- INK:** Valores de 0 a 7
- BRIGHT:** Valores de 0 a 1
- FLASH:** Valores de 0 a 1

El número de combinaciones posibles es de 256 lo cual podremos comprobarlo a través de un último programa.



```

10 DEN = COMBINACIONES DE COLOR
20 FOR I=0 TO 7: FOR J=0 TO 7: FOR K=0
TO 1: FOR L=0 TO 1
30 GO SUB 701: PRINT I;
40 PRINT J; NEXT K; NEXT J; NEXT I
50 PAUSE 0.1: PAPER = I: INK = J: BRIGHT =
FLASH: GOTO 20
60 STOP
70 PRINT I: INK = BRIGHT: FLASH =
80 PRINT PAPER: GOTO 20: INK = I: J = J: GO
TO 1: PAUSE 0.1
90 RETURN
    
```

EL CODIGO MAQUINA



AMOS ahora a hacer algunas precisiones a cerca de la carga y ejecución de programas en código máquina con el Spectrum, quizás algunos de vosotros ya los sepamos, pero en todo caso no está de más refrescar esos conocimientos.

Una subrutina en código máquina es un conjunto de números (bytes) situados en determinada zona de la memoria, a los cuales se accede mediante la función USR. Así pues, para que un programa que utilice determinada subrutina en C/M (código máquina) funcione correctamente, esta se debe encontrar cargada en la memoria.

Al desconectar nuestro aparato, o al pulsar el botón de reset en el PLUS, la memoria RAM se borra íntegramente. Esto quiere decir, que no es suficiente introducir el código máquina una sola vez, sino siempre que vayamos a cargar el programa que la necesite.

Veamos un ejemplo. Vamos a hacer un pequeño programa en código máquina, realmente, lo único que hace es retornar al BASIC. Así pues, la subrutina va a ser extraordinariamente corta, dado que sólo estará compuesta por el código 201. Este será depositado en la dirección 30000. Para introducir en la memoria tan corto programa, sólo es necesario ejecutar la siguiente instrucción: `POKE 30000,201`. Gracias a ella, si tenemos nuestra mini-subrutina en la dirección 30000. Para comprobar que sigue allí, no tenemos más que preguntar el contenido de esa dirección (`PRINT PEEK 30000`). Bien, ahora llega el momento de la verdad, ejecutaremos la subrutina con `RANDOMIZE USR 30000`.

Casi inmediatamente (el tiempo que el intérprete BASIC tarda en traducir al microprocesador la or-

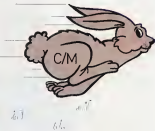


La función USR produce una interrupción temporal del control BASIC hacia una subrutina en código máquina.

den), recibiremos el mensaje `0 OK`. Acabemos de terminar con éxito nuestro primer programa en código máquina, si bien es cierto que no tiene mucha utilidad (simplemente vuelve al BASIC). Ahora bien, él nos va a dar oportunidad de comprobar en la práctica que su efecto desaparece al apagar el ordenador.

Ahora que ya hemos vuelto a encender, ejecute-

La principal ventaja del código máquina sobre el BASIC reside en su extraordinaria velocidad de ejecución.



!

mos de nuevo la instrucción **RANDOMIZE USR 30000**. Esta vez no ha ocurrido lo mismo, y ello se debe a que en la dirección 30000 ya no se encuentra nuestra subrutina en código máquina.

CONCEPTOS IMPORTANTES



Estas dos tarjetas distribuidas entre los lenguajes BASIC y código máquina.

Existen diversas distribuciones entre los lenguajes BASIC y código máquina. Si lo ejecutamos mediante **RUN** o **RUN 10**, que en este caso es lo mismo puesto que 10 es la primera línea de instrucción, cumplirá perfectamente su cometido, sin embargo, ejecutado mediante **RUN 20**, producirá un error del tipo **2 Variable not found**. La traducción de este programa al lenguaje máquina será algo más complicada que un mero código 201, pero no mucho más. Supongamos que como en el caso anterior, deseamos ubicar la subrutina a partir de la dirección 30000, de ser así, la introducción del programa se conseguirá mediante la siguiente línea de instrucción: **POKE 30000,62: POKE 30001,199: POKE 30002,50: POKE 30003,0: POKE 30004,55: POKE 30005,201**.

Ejecutando este mini-programa con **RANDOMIZE USR 30000**, podremos ver que se obtiene un efecto idéntico al observado con la ejecución del último listado BASIC, es decir, la aparición de un cuadrado en intermitencia (negro y blanco), en la posición superior izquierda de la pantalla. Pero las similitudes entre estos dos programas, pese a estar escritos en lenguajes diferentes, no acaban ahí.

Siguiendo un sistema muy parecido al que utilizamos para ejecutar por segunda vez el programa BASIC, vamos a lanzar el código máquina, no desde su dirección inicial (30000), sino desde una más adelantada: **RANDOMIZE USR 30001**. El efecto ha sido aun más desastroso que el que se obtenía al ejecutar **RUN 20**, y ello se debe exclusivamente, que al no encontrarnos bajo el control del firmware, carecemos de un mecanismo de emisión de errores, como el de **2 Variable not found**, para advertirnos de que algo no marcha del todo bien.

Así pues, el igual que hay que tener cuidado para ejecutar un programa BASIC desde el lugar adecuado, aún más debemos tenerlo, para ejecutar el código máquina desde su punto exacto de inicio.

Un programa en código máquina sólo puede ser ejecutado desde la dirección de memoria destinada a tal fin.

*

Un programa de máquina realizable, cuando cualquiera que sea la dirección de memoria en la que se ejecuta, funciona correctamente siempre y cuando sea ejecutado desde la dirección adecuada.

*

La reutilizabilidad no es un hecho común y precisamente sólo se da en rutinas relativamente cortas en longitud y complejidad.

*

La velocidad de ejecución del código máquina es extraordinariamente superior a la del BASIC y llega a ser comparable de velocidad mayor.

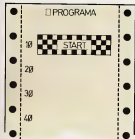
Estudiaremos mediante un nuevo ejemplo un concepto muy importante. Al igual que los programas BASIC, el código máquina tiene su adecuado punto de inicio, y si queremos que funcione correctamente, tenemos que ejecutarlo (USR) a partir de éste. Observemos el siguiente programa BASIC:

```
10 LET A=199
20 POKE 22528,A
```

Su misión es encender en la esquina superior izquierda de la pantalla, un cuadrado de color negro.

La instrucción POKE se utiliza para la introducción por el teclado de programas en código máquina.





Los programas, tanto en BASIC como en código máquina, deben ejecutarse desde la línea para la que están preparados.

REUBICABILIDAD

Ahora vamos a utilizar un nuevo concepto: la REUBICABILIDAD, aplicable tanto al BASIC como al C/M (código máquina), pero que como en el último ejemplo citado, tiene aún más importancia al tratarse en lenguaje máquina, puesto que es más susceptible de producir «desastres». Un programa se dice REUBICABLE cuando cual quiera que sea su situación en la memoria, o su numeración de referencias, funcionará igualmente. Con un ejemplo quedará más clara esta idea. En el caso del BASIC, la numeración de líneas de instrucción, por tanto, un programa BASIC que funcione correctamente cualquiera que sean los números de línea que se utilicen, siempre y cuando se respete el orden de ejecución, es reubicable. El programa del cuadrado intermitente ejemplo es reubicable:

```
10 LET A=199
20 POKE 22628,A
```

El programa funcionará correctamente cualquiera que sean los números de línea que se utilicen,

respetando lógicamente dos normas evidentes: las líneas han de encontrarse en el mismo orden, y el programa se debe ejecutar desde el punto adecuado (en nuestro caso la primera instrucción). Como prueba de ello, tenemos el siguiente programa, que habrá de ejecutarse mediante RUN 50 (o simplemente RUN):

```
50 LET A=199
90 POKE 22628,A
```

Al igual que para el BASIC los puntos de referencia son los números de instrucción, para el lenguaje máquina lo son las direcciones de memoria donde se encuentra. Vamos a cambiar de lugar nuestro programa en M/C (máquina de código máquina) del cuadrado intermitente. POKE 31000,62: POKE 31001,199: POKE 31002,50: POKE 31003,0. POKE 31004,88: POKE 31005,201.

Como recordaremos, este programa se ejecuta desde la primera dirección, que en este caso ya no es 30000, sino 31000. Efectivamente, con RANDOMIZE USR 31000 seguirá funcionando a lo perfección. Este programa es por tanto REUBICABLE también en código máquina. Sin embargo, esta propiedad no depende de la misión que cumple el programa, sino de la manera en que es llevado a cabo. Así pues, podemos encontrar programas que cumplan un mismo objetivo en BASIC y en código máquina, y que sin embargo en un caso sean reubicables y en otro no. De todas formas, la reubicabilidad es un factor de comodidad muy a tener en cuenta, pero lamentablemente no es posible en todas las ocasiones.

Los programas que funcionan cualquiera que sea la referencia que se les otorga se denominan reubicables.



BITS

Por corto que sea un programa en lenguaje máquina, por ejemplo un mini código de sistema (201), siempre tendrá en ejecución un tiempo máximo, debido al trabajo del intérprete BASIC en ejecutar el USR BASIC, y manejar los parámetros de retorno según lo indicado la palabra clave que precede a la función.



Para hacernos una idea de la velocidad del lenguaje máquina basta con saber que la instrucción más básica del Z 80 tarda en ejecutarse 4 usados (1) en el caso del Spectrum 4T: 3500000 usg = 0.00000314285741 usg, y la más complicada, 231 = 231/3500000 usg = 0.00006657142857. Es decir, la velocidad de ejecución por código oscila entre menos de 2 microsegundos (instrucciones de sistema) y menos de 7 microsegundos.

En el siguiente ejemplo, veremos un programa BASIC no reubicable

```
10 LET A=199
20 POKE 22529,A
30 GO TO 50
40 NEW
50 STOP
```

Este programa se ejecutará perfectamente desde la primera instrucción, pero sin embargo, sólo con hacer un pequeño cambio en los números de línea, puede que tenga resultados totalmente diferentes, aún siendo ejecutado desde el punto adecuado (la primera de sus líneas).

```
10 LET A=199
20 POKE 22529,A
30 GO TO 50
50 NEW
55 STOP
```

Esto mismo sucede con el código máquina. Aquellos programas que para funcionar correctamente, se realizan en ellos ningún cambio, deben estar situados en determinado punto de la memoria se denominan NO REUBICABLES

LA FUNCION USR

Para la ejecución de una subrutina en código máquina, recurrimos a la función BASIC USR seguida de la dirección de ejecución de la misma. Como ya sabemos, las funciones BASIC tienen que ir precedidas en una palabra clave para que sean admitidas por el comprobador de sintaxis. Así por ejemplo, si quieres ejecutar una subrutina desde la dirección 30000, no puedes introducir directamente USR 30000, sino que tendrás que emplear PRINT USR 30000, RANDOMIZE USR 30000, o cualquier otra palabra clave. Otro de los características de las funciones BASIC es devolver un valor determinado. En el caso de USR, el resultado corresponde con el valor de los registros B y C del código máquina. Como recordemos del capítulo anterior, los registros son algo así como las «variables» del código máquina.

Por tanto, si utilizamos PRINT USR ... ó la vari-



La función USR siempre debe ir precedida de una palabra clave.

de de la subrutina encontremos un número escrito en la pantalla, puesto que la palabra clave utilizada como «trampolín» de USR ha sido PRINT. Para solucionar este problema, normalmente se utiliza la palabra clave RANDOMIZE, dado que no produce ningún efecto en la pantalla.

Pese a todo, este último sistema puede ocasionar ciertos problemas en programas que utilicen la generación de números aleatorios (RND). El método más eficaz para que la llamada al código máquina no provoque problemas, es el almacenamiento en una variable del resultado devuelto LET <variable> = USR ... Por ejemplo, LET A=USR 30000.



El sistema ideal para utilizar USR es almacenar su resultado de retorno en una variable.



!/

Derrente la ejecución de un programa en código máquina puede ser que se produzca una detención con un extraño mensaje del Spectrum, esto será debido a que por un error en nuestra subrutina, la ejecución ha pasado a alguna zona de la ROM (Vermas).

*

La unidad de medida de tiempo de ejecución para el código máquina es el ESTAD [T]. Este equivale a una partida por la frecuencia de reloj del microprocesador. En el caso concreto del Z 80 que incorpora nuestro Spectrum, la frecuencia de reloj es de 3 500 000 Hz su ócio T=1/3500000 seg.

BLACK JACK



¡MOS creado para nuestros incondicionales lectores el RUN'S CASINO. Como acabamos de inaugurar, andamos escasos deroupéras que mantengan la totalidad de nuestras mesas, pero nuestro fiel amigo Spectrum se ha presentado voluntario para atender la mesa de Black Jack. Muchos de nosotros hemos oído hablar del Black Jack, pero desconocemos sus reglas de juego. Por ésto, a continuación aparece una pequeña sinopsis de dichas normas, para que así lleguemos a ser unos empedernidos y expertos jugadores de Black Jack.

Antes de nada hemos de advertir que las reglas de nuestro Black Jack no son exactamente las tradicionales del casino. Su mayor mejora consiste en que no se juega con pesetas, sino con una nueva moneda: el RUN DOLAR; un considerable ahorro para aquellos que no tienen buena fortuna con las cartas.

Por otra parte, aunque algunos desconozcamos el juego, su mecanismo no nos será del todo ajeno, puesto que es muy similar al populatísimo siete y medio.

La mejor jugada es el Black Jack, la cual se premia con el doble de la apuesta.



Las valores de las naipes corresponden con los números de las cartas, a excepción de las figuras, que tienen valor diez, y el as, el cual puede valijarse indistintamente los valores once o uno.

BITS

REGLAS DEL JUEGO

El Black Jack se juega con una baraja francesa (diamantes, corazones, tréboles y picas). El valor numérico de las cartas, es el impreso en las mismas, excepto en el as (A), el jack (J), la dama (Q) y el rey (K), cuyos valores respectivos son los siguientes: el as puede asumir los valores 1 u 11 según nos convenga, aunque en nuestro caso particular va a ser el Spectrum quien dé el valor que se le antoje dentro de los dos citados. El valor de las tres cartas restantes (J,Q,K) es 10.

Al comenzar el juego, el ordenador nos reparte una carta, y se despacha a sí mismo otra, ambas boca arriba. Aunque nuestros naipes aparezcan al descubierto en la pantalla, no debemos preocuparnos: el Spectrum es honrado a carta cabal, y no nos los mirará ni lo más mínimo. El objetivo del juego es quedar los más próximos

La protección del programa debe llevarse a cabo mediante el comando **SAVE "BLACKJACK"**. Si volvemos el sistema de autoselección, esta ha de comenzar a partir de la línea 1 (**SAVE "BLACK-JACK" LINE 1**).



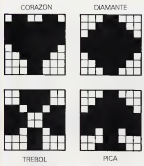
Al introducir el programa, no olvidemos sustituir los caracteres subrayados por los gráficos de las fichas correspondientes



En este programa, la duplicación de los caracteres se obtiene mediante una subrutina en BASIC la cual accede a la dirección de la ROM donde se encuentran almacenados los códigos que definen los caracteres del Spectrum



El premio por obtener la jugada de Black Jack, va en nuestro juego, el doble de lo apostado



Cuatro las cías los gráficos respectivamente definidos para representar las pajas de la bodega.

posible a veintín puntos, pero sin sobrepasarlo, puesto que en ese momento quedamos descalificados. Para conseguirlos, a la vista de nuestras cartas, y de la que le vemos al ordenador, podemos pedir un saque más, hasta que nos planteemos en determinada puntuación. El número máximo de cartas que podemos solicitar es cinco. Una vez llegados a este punto, será al Spectrum al que le toque jugar, aunque en su caso el abanico de posibilidades es mucho más restringido tendrá que ir sumando puntos hasta un máximo de 17, estando obligado a plantarse en el momento en que alcance o supere dicha cantidad. A continuación se procede a la comprobación de cual de los dos oponentes ha obtenido la mejor jugada. En este aspecto, el Spectrum se guarda dos bases bastante importantes: la primera es que en caso de empate a puntos, siempre se declara vencedor el crupier, y lo segundo, que si hemos sobrepasado la puntuación de veintinueve, habremos perdido la «mano», sin tener para nada en cuenta si a él tampoco le acompañó la suerte. Por último, existe una jugada que supera a todas, incluso a la de obtener veintín puntos. Esta maravillosa combinación de cartas se denomina Black Jack, y consiste en obtener de entrada, es decir, con las dos primeras cartas, una puntuación de veintinueve (un as y una carta de valor diez). Así pues, podemos resumir las reglas del juego en los cinco puntos siguientes:

1. Se consigue Black Jack cuando con nuestras dos primeras cartas obtenemos veintín puntos.
2. Si una vez pedidos dos cartas no hemos conseguido Black Jack, seguiremos sumando puntos, intentando acercarnos lo más posible al valor 21

3. Podemos plantarnos en cualquier momento del juego, siempre que no hayamos sobrepasado los veintín puntos, en cuyo caso ya no tendríamos sentido seguir la mano.

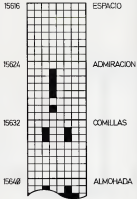
4. Si el Spectrum iguala nuestra puntuación habremos perdido ese bazo, y en consecuencia lo que hubiésemos apostado.

5. Si la puntuación del Spectrum es 16 o menor, nuestro crupier personal tiene la obligación de pedir una nueva carta. Por el contrario, si la suma de los puntos obtenidos por el Spectrum es 17 o superior, se plantará, aunque con ello sepa que perdió la jugada.

La arquitectura del programa es completamente estructurada y sencilla de comprender, a excepción quizá de la subrutina que duplica el tamaño de los caracteres ingresados en los saques. Esta parte del programa se encuentra en la línea 8000, y a breves rasgos, funciona de la manera que a continuación explicamos.

La subrutina busca a partir de la dirección de memoria 15516 (ROM), el carácter elegido por el jugador para su saque. A partir de dicha dirección, se hayan almacenados todos los caracteres, tanto numéricos como alfabéticos, de nues-

A partir de la dirección de memoria 15616 se encuentra el área de generación de caracteres





tro Spectrum. Una vez encontrado el carácter oportuno, el programa duplica su tamaño y lo ubica en el área destinada a los gráficos de usuario para su posterior uso.

Este método es algo lento. Podíamos haber utilizado la subrutina de caracteres gigantes de Psion Computers, y evitarlos así el aumento de caracteres, pero... como bien es sabido: «En la variación está el gusto». Nuestro objetivo ha sido además, ilustrar otra técnica con la que poder conseguir un efecto similar.

Para la introducción del programa, tengamos en

cuenta que los caracteres subrayados corresponden a los gráficos de las teclas afectadas, así por ejemplo, A equivale al carácter gráfico de la A, es decir, el que se obtiene pulsando la tecla A cuando nos encontramos en el modo GRAPHICS (cursor G). Por otra parte, los caracteres doblemente subrayados (este hecho sólo se da con números), corresponden a los gráficos cambiados de los números afectados, los cuales se obtienen de forma similar a los anteriores, pero pulsando además CAPS SHIFT.

Para finalizar, resta advertir que es una buena

